Gitintro

9 september 2013

Sammanfattning

Detta är ett introduktionsdokument som i snabba drag beskriver hur man använder Git. Dokumentet består utav två delar. Den första delen beskriver hur man installerar och konfigurerar Git samt vissa tips på hur man bör använda det medans den andra delen beskriver arbetsflödet för att införa ett litet stycke med ny funktionalitet. Beskrivningen av arbetsflödet bör ses som regler för exakt hur vi ska använda Git och om man inte vet vad man pysslar med så bör man inte frångå dom.

Del I

Införskaffa Git

Skapa ett konto på github.com.

För windows så finns det viss nödvändig mjukvara som även om det ej är strikt nödvändigt är starkt rekommenderad att installera och konfigurera. I denna guide förutsätts att all mjukvara är installerad och länkarna går till rekommenderade versionen (2013-09-09).

Notationsvarning: hemmapp, profilmapp och ~ syftar allihopa på din användarmapp (vanligtvis "C:\Users\username") du kan komma till din hemmap genom att skriva %userprofile% i kör prompten.

1 Mjukvara att installera

Akta dig för crapware i installationsprogrammen.

Putty http://the.earth.li/~sgtatham/putty/latest/x86/putty-0.63-installer.exe

Tortoisegit http://tortoisegit.googlecode.com/files/TortoiseGit-1.8.5.0-64bit.msi

Notepad++ http://download.tuxfamily.org/notepadplus/6.4.5/npp.6.4.5.Installer.exe

Msysgit https://code.google.com/p/msysgit/downloads/detail?name=Git-1.8.3-preview20130601.exe&can=2&q=

Console2 http://sourceforge.net/projects/console/files/latest/download?source=files

Putty

Installera med normala inställningar, när den har kört färdigt så ska du generera en ssh-nyckel med puttygen. Starta puttygen och följ instruktionerna för att generera en ssh-nyckel. Ssh-nyckeln fungerar som ett lösenord som är sparat lokalt på datorn. Spara den färdiga nyckeln som puttykey.ppk i din hemmapp. Kopiera in den publika delen av nyckeln till github.

Tortoisegit

Tortoisegit ger diverse verktyg för att hantera git med ett behändigt gui. Vi kommer främst använda git via git bash men flera tortoisegit innehåller vissa verktyg som kommer vara ovärdeliga.

Notepad++

Editor med syntaxmarkeringar och annat användbart.

Msysgit

Git för windows, installera så att den använder putty (plink) och "check out windows style check in unix style". När den frågar vilken sorts högerklicksmeny som ska användas (context-menu) välj "ingen".

Skapa en mapp som heter git i din hemmapp. Starta notepad++ och skapa följande filer i din hemmapp:

npp.sh

```
#!/bin/sh
"C:/Program Files (x86)/Notepad++/notepad++.exe" -multiInst "$*"
```

.gitconfig

```
[alias]
        st = status
        co = checkout
        lol = log --graph --pretty=oneline --abbrev-commit
        ls = log --graph --pretty=short --abbrev-commit
[core]
        editor = ~/npp.sh
[user]
        email = name@example.com
        name = Förnamn Efternamn
```

Console2

Ingen installer utan en zippad mapp, extrahera så att det ligger i hemmappen (~/Console2). Starta Console2.exe och tryck på edit->settings; tabs; tryck på add och mata in följande:

Title Git Bash

Icon C:\Program Files (x86)\Git\etc\git.ico

Shell C:\Windows\SysWOW64\cmd.exe /c ""C:\Program Files (x86)\Git\bin\sh.exe" –login -i"

Startup-dir %userprofile%/git

Spara en genväg till Console2 (förslagsvis genom att pinna den på aktivitetsfältet)

Del II
Arbetsflöde

Utgående från att du har ett git, filstrukturen från ovan och allting är konfigurerat. Man bör relativt ofta skriva git lol (git log one line) för att få en graf som beskriver hur trädet ser ut, att göra detta hjälper dig få översikt över vad det faktiskt är du gör. Du kan också använda tortoisegits funktionalitet (högerklick->show log) som också ger en bra överblick.

2 Klona repot

Man måste börja med att klona repot, dvs ladda ner det som gjort hittils. Från git mappen i din hemmap skriv följande

```
git clone git@github.com:matni796/robot-security-tsbb11.git tsbb11
```

Alternativt clone via tortoise-git (högerklick, clone repository)

3 Påbörja ny funktionalitet

Det första man gör innan man påbörjar ny kodning är att göra en ny branch. En branch är en gren där man kan utföra förändringar utan att vara rädd för att man förstör för andra. Alla förändringar som inte är triviala bör göras i en ny branch.

```
git checkout -b funktionalitets-beskrivning develop
```

Du kan, men bör ej, göra det här via tortoise.

4 Skriv funktionaliteten

Efter det så börjar man koda. Då och då så kommer man till ett naturligt tillfälle att commita (minst en gång per arbetstillfälle) och lägger till saker som är färdiga. Det görs genom att lägga till de filer som är redo och sedan commita:

```
git status
git add fil1
git add fil2
git commit
```

När man skriver git commit så ska man skriva ett commitmeddelande, commitmeddelande skriva på följande sätt:

```
ISSUE-ID: Kortfattad mening som beskriver ändringen

Längre mening som beskriver vad man gjort och hur man gjort det. Det är
rimligt att skriva en eller ett par paragrafer här men något som man
tyvärr måste hålla reda på själv är radbrott. Raderna får absolut inte
vara längre än 72 tecken.

Radbrottet mellan första raden och resten av texten är obligatorisk.

Om det är många filer som manipulerats så kan man göra listor som
beskriver förändringen:

 - kod.cpp Kommenterade några okommenterade funktioner
 - kod2.cpp Lade till grundfunktionalitet för något.

Och så vidare.
```

Det är extremt viktigt att första raden på commit-meddelandet följer den här strukturen. Första raden tolkas av verktyg som en identifierare för hela committen och får absolut inte vara oseriöst. Inga Hej eller blabla eller något annat skräp.

Om man har många filer som man manipulerat men inte vill commita så rekommenderar jag starkt att använda tortoise för det här, då kan man behändigt använda en check-lista och även få lite hjälp med komplettering av filnamnen (högerklick->commit, så får du välja vilka filer du vill lägga till i commiten).

5 Uppdatera ditt git träd

Efter att man gjort en commit så har man en bra möjlighet att uppdatera sitt träd, detta bör göras minst en gång per arbetstillfälle (rimligtvis precis innan man börjar arbeta för dagen) detta innebär att man hämtar de förändringar som gjorts av resterande medlemmar.

```
# hämta förändringar
```

```
git fetch origin
# arbeta med develop-grenen
git checkout develop
# spola fram din lokala develop till den nyare versionen
git merge --ff-only origin/develop
# tillbaka till din egen gren
git checkout funktionalitets-beskrivning
```

Bör ej göras i tortoise.

6 Applicera dina förändringar ovanpå den senaste versionen

Efter att du hämtat förändringar så ser man i loggen att alla dina förändringar är applicerade på en äldre version av koden. Detta löser man med rebase, som applicerar dina förändringar ovanpå den senaste versionen.

```
git rebase develop
```

Bör ej göras i tortoise.

7 Färdig med funktionalitet

När du är färdig med funktionaliteten så bör du merga in den tillbaks till develop.

```
git checkout develop                          # arbeta i develop
git merge --no-ff funktionalitets-beskrivning # stoppa in dina förändringar i develop
git branch -d funktionalitets-beskrivning     # släng den gamla grenen
git push origin develop                       # skicka förändringar till servern
```